ASSIGNMENT 2

INTRODUCTION TO NLP

SRIHARSHITHA BONDUGULA

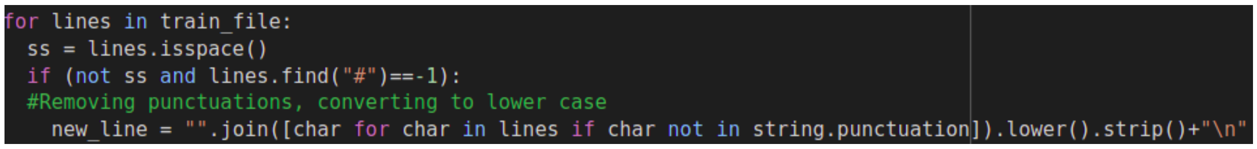
2018111013 NEURAL LM

Neural Language model (LSTM based)

Preprocessing

The first part of the language modeling is to clean the text. As a part of cleaning the text, the following operations are done on the sentences using regex.

1) The dataset given contained many paragraphs and there is a heading for each paragraph. I have removed the headings using regex. The following snippet is used to clean the text.

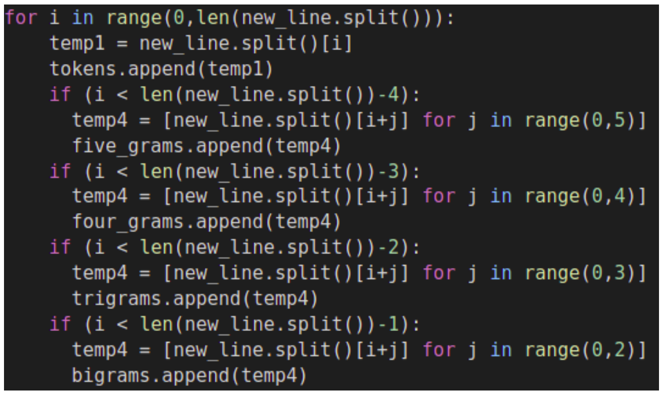


```
for lines in train_file:
  ss = lines.isspace()
  if (not ss and lines.find("#")==-1):
  #Removing punctuations, converting to lower case
    new_line = "".join([char for char in lines if char not in string.punctuation]).lower().strip()+"\n"
```

2) Punctuations are removed using string.punctuation() library.
3) Each line is then converted into lower case.

Tokenisation and obtaining n-grams

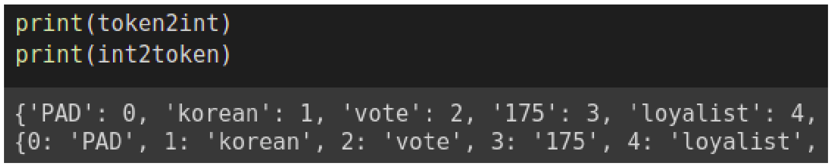
The preprocessed line obtained is then split and all the tokens are collected. The LSTM model I have is a 5-gram model. So, I have collected all the bigrams, trigrams, 4-grams, and 5-grams. Following is the code snippet used for the same.



```
for i in range(0,len(new_line.split())):
    temp1 = new_line.split()[i]
    tokens.append(temp1)
    if (i < len(new_line.split())-4):
      temp4 = [new_line.split()[i+j] for j in range(0,5)]
      five_grams.append(temp4)
    if (i < len(new_line.split())-3):
      temp4 = [new_line.split()[i+j] for j in range(0,4)]
      four_grams.append(temp4)
    if (i < len(new_line.split())-2):
      temp4 = [new_line.split()[i+j] for j in range(0,3)]
      trigrams.append(temp4)
    if (i < len(new_line.split())-1):
      temp4 = [new_line.split()[i+j] for j in range(0,2)]
      bigrams.append(temp4)
```

Int2token and Token2int dictionaries

We cannot feed sentences to neural networks in the form of text. To solve this problem each word/token is assigned an id (integer) and it is stored in a dictionary called token2int. The network gives an integer as an output which will the id of some token. This should hence be converted into the token for which we need an int2token dictionary. The following are the 2 dictionaries:



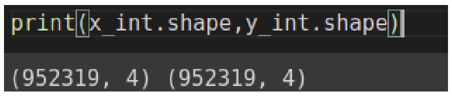
```
print(token2int)
print(int2token)

{'PAD': 0, 'korean': 1, 'vote': 2, '175': 3, 'loyalist': 4,
{0: 'PAD', 1: 'korean', 2: 'vote', 3: '175', 4: 'loyalist',
```

Considering most_common words as vocabulary and others as 'unk'

I have calculated the number of tokens with a frequency of 1 by using Collections.counter(). I have considered all these words as 'unk' tokens.

Obtaining input and target sequences and feeding them to the neural network

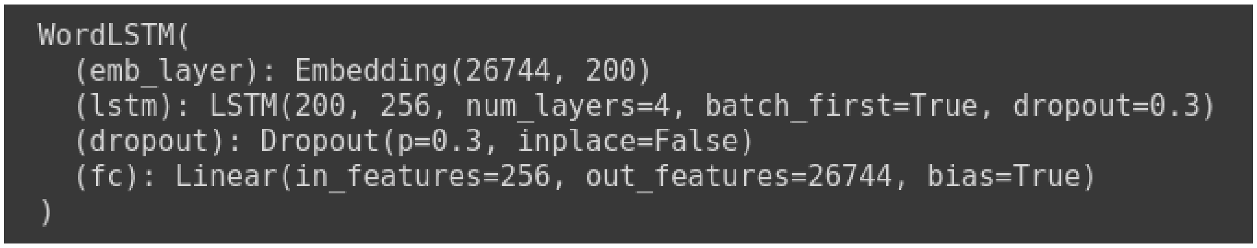
Input sequences and target sequences are obtained for all the grams. These are then converted into integer sequences. Sequences of length less than 5 are padded. Now, all the sequences are of uniform length and can be fed to train the model.



```
print(x_int.shape,y_int.shape)

(952319, 4) (952319, 4)
```

Model

The following is the model that I have built and used;



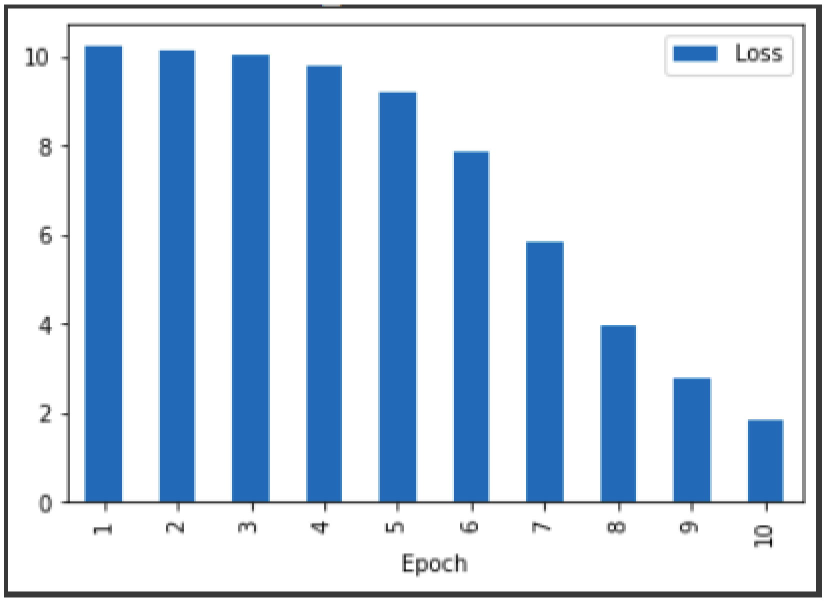
```
WordLSTM(
  (emb_layer): Embedding(26744, 200)
  (lstm): LSTM(200, 256, num_layers=4, batch_first=True, dropout=0.3)
  (dropout): Dropout(p=0.3, inplace=False)
  (fc): Linear(in_features=256, out_features=26744, bias=True)
)
```

Training

I have trained the model with the input and target sequences constructed for 10 epochs.
Parameters used are;

Number of epochs:	10
Batch size:	32
Learning rate:	0.001
Loss function:	CrossEntropyLoss()
Optimiser:	Adam

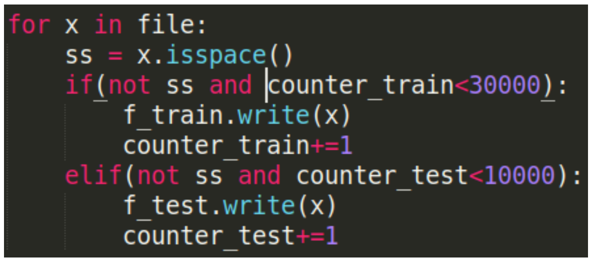
Loss per epoch is calculated and the following is the graph of the same.



Loss per epoch is decreasing and this clearly indicates that the model is learning and it is converging.

Train-Test division

I have considered the first 30K sentences as the train set and the following 10K sentences as the test set. Following is the code snippet I used to make train and test sets.



```
for x in file:
    ss = x.isspace()
    if(not ss and counter_train<30000):
        f_train.write(x)
        counter_train+=1
    elif(not ss and counter_test<10000):
        f_test.write(x)
        counter_test+=1
```

I have not used any validation set in my model.

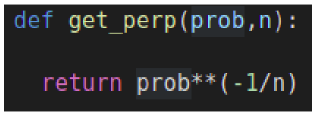
Predicting probabilities of sentences

I have used the chain rule to predict the probability of a sequence. The softmax layer of the model and the concept of hidden state in LSTMs play a major role in this.

These predicted probabilities are further used to predict the perplexities of sentences. The formula used to calculate perplexity is as follows;

$$\begin{aligned} \mathrm{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}} \end{aligned}$$

w1w2...wN is the sequences. (N=5 in my LSTM model). P(w1w2...wN) is expanded using the chain rule.



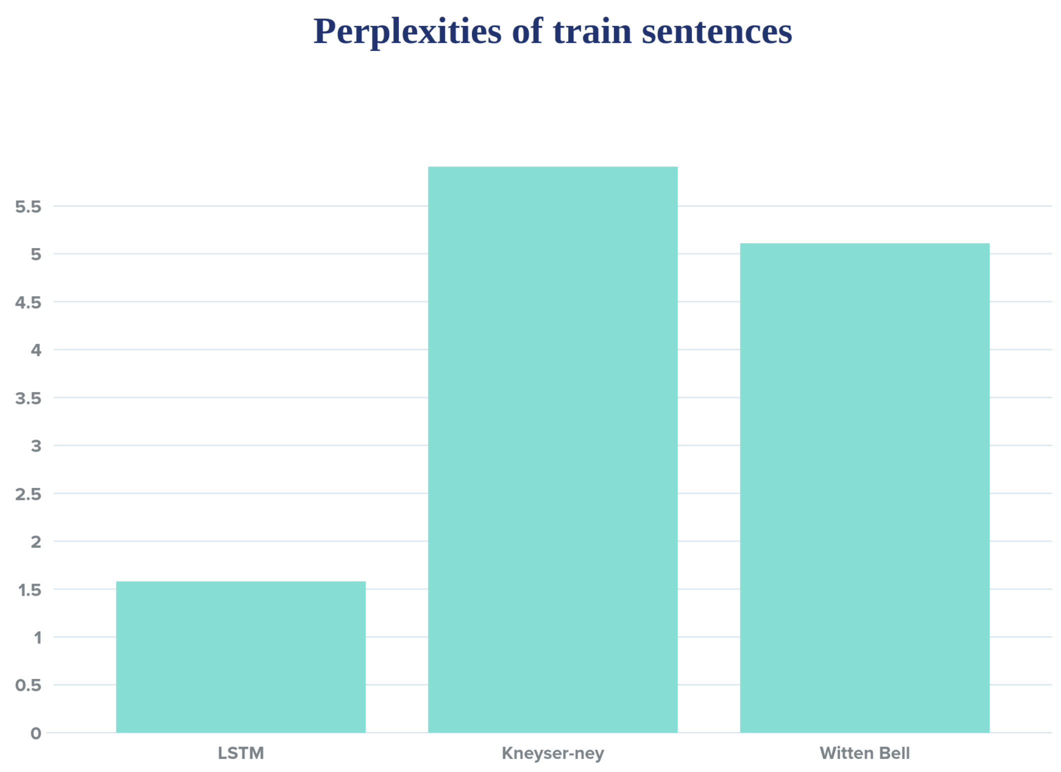
```
def get_perp(prob,n):

  return prob**(-1/n)
```

Results

- Trained all the language models on 30K sentences (trained on the same dataset)
- Tested statistical LMs on 1000 test sentences. And the average printed below.
- Tested statistical LMs on 1000 train sentences as well. And the average is printed below.
- Tested Neural LM, i.e LSTM on 10K test sentences. And the average is printed below.
- Tested Neural LM on 30K train sentences and printed the average in the table below.

	Train	Test
Kneyser-ney	5.910104796669125	142.6039750276732
Witten-Bell	5.11449955654094	874.8967097019614
LSTM	1.5709781269053038	1.606463082173958

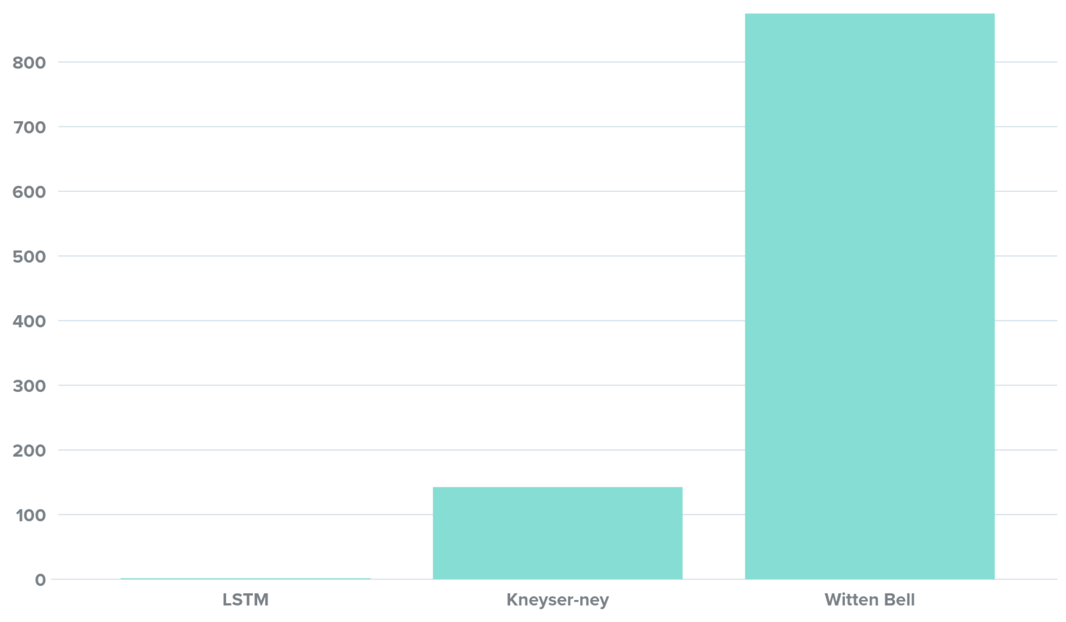
Visualizations & Analysis



- From the above graph, the average perplexity of the training set given by the Kneyser-ney model is higher than that given by the Witten-bell model. Thus, the probabilities given by the Kneyser-ney model must have been lesser than those given by the Witten-bell model. This tells that Witten-bell is more conservative and deducts lesser value from probabilities of seen sentences, i.e the sentences in the training set. Whereas the penalty imposed by Kneyser-ney on seen sentences is higher.

- The average train perplexity of the LSTM model is lesser than both the statistical models. This tells that the probabilities of the train sentences according to the LSTM model are higher. Thus, the LSTM model does not deduct probabilities from seen sentences. It just predicts the probability based on what it has learned.

Perplexities of test sentences



- From the graph above, the average perplexity of the test set given by the Witten-bell is higher than that given by Kneyser-Ney. This means that the probabilities of test sentences given by Witten-bell are lesser than those given by Kneyser-Ney. This proves that Witten-bell is conservative and takes lesser probabilities from seen sentences and hence gives lesser probabilities to unseen sentences. Comparatively, the penalty on seen sentences is more when the Kneyser-ney model is used and hence probabilities for them are high.

- LSTM model does not deduct probabilities of train sentences. It predicts the probabilities of test sentences by learning from train sentences. It gives almost the same perplexities to both seen and unseen sentences. (This is because we used an 'unk' token for the words that are rarely seen.)

- Test and train perplexities of the LSTM model are less than those of the statistical models.

- The lower the perplexity on the test corpus, the better is the model.

- Hence, the LSTM model outperforms the other 2 statistical models. From the perplexity scores, it is quite clear that the performance of a Neural LM is remarkable.